Bot-IMG: A framework for image-based detection of Android botnets using machine learning

Suleiman Y. Yerima* and Abul Bashar†

*Cyber Technology Institute, Faculty of Computing, Engineering and Media
De Montfort University, Leicester, United Kingdom

†College of Computer Engineering & Sciences
Prince Mohammad Bin Fahd University, Al-Khobar, Kingdom of Saudi Arabia

Corresponding email: syerima@dmu.ac.uk

***Abstract*—To enable more effective mitigation of Android botnets, image-based detection approaches offer great promise. Such image-based or visualization methods provide detection solutions that are less reliant on hand-engineered features which require domain knowledge. In this paper we propose Bot-IMG, a framework for visualization and image-based detection of Android botnets using machine learning. Furthermore, we evaluated the efficacy of Bot-IMG framework using the ISCX botnet dataset. In particular, we implement an image-based detection method using Histogram of Oriented Gradients (HOG) as feature descriptors within the framework, and utilized Autoencoders in conjunction with traditional machine learning classifiers. From the experiments performed, we obtained up to 95.3% classification accuracy using train-test split of 80:20 and 93.1% classification accuracy with 10-fold cross validation.**

***Index Terms*—Botnet detection; Image processing; Histogram of Oriented Gradients; Machine learning; Autoencoder; Android Malware**

I. INTRODUCTION

The McAfee mobile threat reports of recent years have shown that mobile malware is growing, with majority still targeting Android-based systems [1]. Android malware has become widespread due to its unrestricted distribution via various sources, and the ability of users to permit unverified apps to be installed on their Android devices. Moreover, many app sources lack any process or mechanism to check apps for malware before they are downloaded onto the user devices.

As mobile devices, especially smartphones, tend to be always on and always connected, they provide a suitable platform to operate botnets. Mobile botnets are a group of compromised mobile devices that are remotely controlled by botmasters using command and control channels. Numerous Android malware families that infect mobile devices and incorporate them into botnets have been discovered. These botnets are used for various attacks such as Distributed Denial of Service (DDoS), spam distribution, phishing attacks, click fraud, credentials stuffing, etc.

Some of the earliest known Android botnet include DroidDream and Geinimi, which were distributed through trojanized game apps, compromising Android devices and turning them into bots. A more recent family called Chamois, distributed though Google play and third-party app stores, infected over 20.8 million Android devices between November 2017 and March 2018. Chamois evolved to become more sophisticated and the creators of the botnet countered initial eradication campaigns by bundling Chamois into a fake payment solution for device manufacturers, and fake advertising SDK for developers [2] [3] [4].

Mobile botnets are being increasingly used for credential stuffing attacks where list of usernames and password gathered from breaches are used to attempt to login to targeted websites [5]. The aim could be to hijack identity, gather information or steal money and/or goods. A study by Imperva on mobile botnet activity showed that 5.8 million bot-infected mobile devices were used to attack websites and apps for financial gain over a 45-day period on six major cellular networks [6]. Unlike DDoS attacks that are characterized by high volume, high frequency network traffic, credential stuffing attacks and attacks on websites are low volume and low frequency. This makes such attacks challenging to detect using traditional network intrusion detection systems. Thus, complementary approaches to network-based detection are needed to strengthen defence against mobile botnet infection and attacks.

Moreover, as Android botnets evolve to become more sophisticated, successful eradication will require more effective detection approaches than the existing solutions. Researchers have proposed various machine learning based schemes to enable automated detection of Android botnets. For example [7] and [8] proposed an approach based on API calls, Permissions, Intents and Commands with deep learning for

The research carried out in this paper is supported by the 2021 Cybersecurity research grant from the Cybersecurity Center at Prince Mohammad Bin Fahd University, Al-Khobar, Saudi Arabia.

the detection of Android botnets. Similarly, [9] used machine learning to detect Android botnets based on permissions and their protection levels. Anwar et al. [10] also used permissions, broadcast receivers, MD5, and background services as features to build machine learning classifiers to detect botnet attacks. These existing Android botnet detection techniques rely on hand-engineered features which requires domain knowledge or expertise to extract them. Also, as the Android platforms evolve, many of these hand-engineered features may become deprecated or obsolete and the entire feature extraction process may need to be re-engineered.

To overcome the limitations of hand-engineered features, image-based or visualization approaches offer a more effective alternative. It reduces reliance on (Android specific) domain expertise, and the pre-processing/feature extraction process would require little or no modification to adapt to changes in the platform. Thus, image-based detection enables long-term efficiency and cost-effectiveness compared to hand-engineered features. Therefore, in this paper we propose a framework for image-based detection of Android botnets, called Bot-IMG. Bot-IMG is designed to enable machine learning based detection of Android botnets without the need to extract and utilize 'hand-engineered' features. Instead, Bot-IMG provides image representations of Android applications which is used as the basis to build the machine learning detection models.

This paper is organized as follows: In section II, we present the Bot-IMG framework and describe its various components. In section III, Bot-IMG is used to develop a botnet detection solution based on Histogram of Oriented Gradients (HOG). In section IV, we present a study to evaluate the Bot-IMG framework using the ISCX botnet dataset. In section V, we discuss related work. Finally, in section VI, we present conclusions and future work.

II. Bot-IMG Framework

Bot-IMG is a framework for visualization and image-based detection of Android botnets. It consists of various components that work together to enable high-accuracy machine learning based models to be built. The framework is designed to also enable unknown applications to be predicted as malicious (botnet) or benign (clean). Thus Bot-IMG can be used as an engine for automated botnet detection as part of a cloud/edge-based or device-based app screening service.

The main subsystems of Bot-IMG are (a) reverse engineering/image generation (b) Feature extraction and processing (c) ML model generation (d) classifying/prediction of unknown application.

Reverse engineering/image generation: This subsystem is used by Bot-IMG to generate grayscale or colour images to represent an application. A single image could be generated or several images could be generated and further processed to generate a 'meta-image'. Basically, the application is decomposed into its various constituent files: Manifest, dex file, resource files, etc. An image of the dex (Dalvik executable) file or Manifest is generated, or a meta-image of the Manifest and dex files can be generated, depending on the use case. An image of the resource file can also be generated and used to check hidden executables. For ML-based classification, the dex file images are the most important since it is the representation of the executable code.

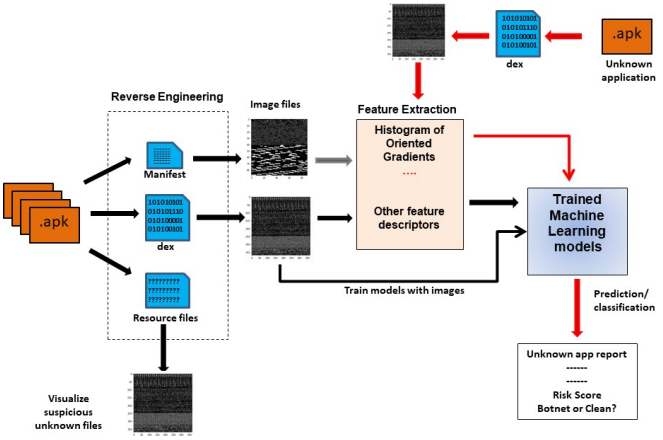

Fig. 1. Overview of the Bot-IMG framework for image-based Android botnet detection

Feature extraction and processing: As shown in Fig. 1, Bot-IMG could be used to build machine learning models directly from the images. Additionally, different types of 'feature descriptors' could be extracted from the images to build the ML models. Presently, the Histogram of Oriented Gradients (HOG) feature descriptor is implemented within the framework. HOG provides a faster method of building image-based ML detection method compared to training models directly from the images. In future, Bot-IMG will incorporate other types of feature descriptors and will explore combining them with HOG descriptors to improve performance.

ML model generation: This aspect of Bot-IMG takes direct images or feature descriptors and uses these to train a ML model for classification of apps to detect botnets. Presently, Bot-IMG can generate classification models based on KNN, SVM, Random Forest, Decision Trees, Extra Trees, and XGBoost.

Classifying/prediction of unknown application: This aspect of Bot-IMG utilizes a trained ML model to predict the class of an unknown app, as shown in Fig. 1. Assuming we are using a HOG feature descriptor-trained SVM model to classify an unknown app (APK). The app will undergo reverse engineering to generate a representative image file to which the HOG feature extractor will then be applied to obtain its HOG descriptor. The trained SVM model will use the descriptor to classify the app as botnet or benign. This could be presented as a form of a 'Risk Score' or statement within a report generated for the unknown application.

III. Using Histogram of Oriented Gradients for Android Botnet Detection

In this section we describe a classification system based on Histogram of Oriented Gradients (HOG) for detecting Android botnets using the Bot-IMG framework. HOG is used within

the feature extraction component of Bot-IMG and provides feature vectors that are used to train the classification models.

A. Histogram of Oriented Gradients (HOG)

HOG [11] is a feature descriptor which is widely used in computer vision and image processing. A feature descriptor is a representation of an image or an image patch/segment that extracts useful information from the image. By using feature descriptors to represent images, the training of various machine learning models becomes more feasible and much less computationally demanding compared to using the images directly.

HOG is a technique that counts the occurrences of gradient orientation in localized portions of an image. It is commonly used for object and edge detection in computer vision applications. For example, pedestrian detection in static mages or human detection in videos [12].

To derive the HOG descriptor vector for an image, the following procedure is performed:

1) The image is divided into smaller R x R connected regions called cells. Each cell contains R x R x 1 pixels for grayscale image (which we are utilizing) or R x R x 3 pixels for colour images.
2) Assuming our image is divided into 8 x 8 pixels per cell (i.e. R =8), a grayscale image will contain 64 pixels per cell. Each gradient of each pixel will be made up of 2 values (magnitude and direction) leading to 128 values per cell.
3) In each cell, the 128 values (of gradient magnitude and direction) are represented using a k-bin histogram which is stored using an array of k numbers. Thus, each cell will be represented by a k-number array. Typically, k is chosen to be 9 which leads to orientations spaced 20 degrees apart (180/9=20). That is, 0, 20, 40, 60 , . . . 160.
4) The k-number arrays or bins correspond to reference 'orientations' or 'angles' to which the orientation of each pixel in the cell is compared. If the pixel orientation is close to the reference orientation, its magnitude is added to the array/bin representing that reference orientation.
5) The k-number arrays or bins correspond to reference 'orientations' or 'angles' to which the orientation of each pixel in the cell is compared. If the pixel orientation is close to the reference orientation, its magnitude is added to the array/bin representing that reference orientation.
6) The binning of the magnitudes according to the orientation produces a histogram of gradient directions for the pixel magnitudes in a cell. Assuming we take k=9, each cell is represented by a 9 x 1 array.
7) The histogram array in each cell is normalized to make the gradients less sensitive to scaling. Normalization is performed using 'blocks' where the number of cells per block are chosen as b x b cells. Assuming b=2, this will give us 4 cells per block. Hence each block will be represented by 4 x 9 x 1 = 36 x 1 vector or array.

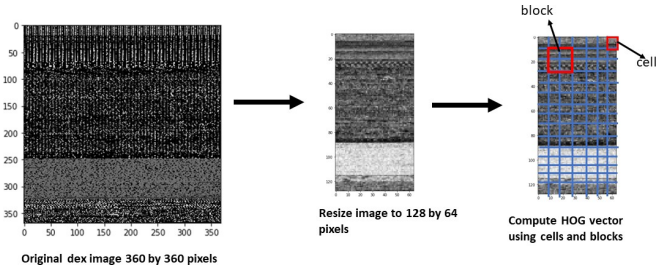

Fig. 2. HOG descriptor extraction from resized dex image

8) The HOG algorithm typically takes an input image size of size 128 x 64, by default. Hence, if we take 8 x 8 pixels per cell, and 2 x 2 cells per block, then there will be 7 horizontal block positions and 15 vertical block positions in the 128 x 64 pixel image. Therefore, the total HOG vector length is calculated by 36 x 7 x 15 = 3,780

Hence, in order to obtain a 3,780 vector HOG descriptor for an image we provide the following parameters: number of orientations, k=9; number of pixels/cell, R=8; number of cells per block, b = 2. The image size must be 128 x 64 pixels, hence a grayscale dex image is resized to 128 x 64 before applying it to the HOG function. The process is illustrated in Fig. 2.

B. Enhanced HOG-based scheme

In this section, we describe an enhanced scheme developed to obtain HOG feature descriptions without the need to resize the images of the dex files. The scheme is motivated by the differences in the sizes of the images that result from extracting grayscale images from apps with different dex file sizes.

The scheme takes a fixed portion of the image, and divides it into N segments, with each segment corresponding to 128 x 64 pixels. The HOG descriptors are the extracted for each segment. We then take a fixed number of HOG descriptors from each segment and concatenate them to form a single vector that will represent the image, and be used for training the machine learning classifiers. The process is illustrated in Fig. 3.

C. Autoencoder implementation

Autoencoder is a type of neural network that can be used to learn a compressed representation of data. Basically, an autoencoder is composed of an encoder and decoder sub-models. Our aim is to use autoencoder to learn a new compressed representation of the HOG descriptors and then use this representation to train different machine learning classifiers. The part of the autoencoder that will learn the new representation is the encoder, while the decoder will attempt to recreate the input from the compressed version provided by the decoder. During the training, the encoder model is updated as the error between the model trained with the original input and the decoder output decreases. When a suitable encoder model that minimizes the error is found, it is saved while the decoder

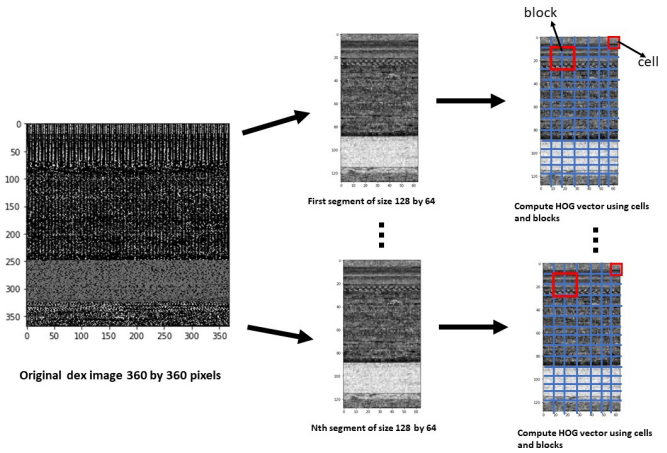

Fig. 3. HOG descriptor extraction from segmentation of dex image

TABLE I
ISCX BOTNET DATASET COMPOSITION

Botnet Family	Number of Samples
Anserverbot	244
Bmaster	6
Droiddream	363
Geinimi	264
Misosms	100
Nickyspy	199
Notcompatible	76
Pjapps	244
Pletor	85
Rootsmart	28
Sandroid	44
Tigerbot	96
Wroba	100
Zitmo	80
Total	**1929**

model is discarded. The saved encoder model contains the new extracted features which are then used as the input for training a machine learning classifier.

To implement the autoencoder, we used the Keras library with Tensorflow backend. Our autoencoder design consists of an input layer, 2 hidden layers of the encoder, a latent or bottleneck layer, 2 hidden layers of the decoder, and an output layer. The encoder and decoder layers are symmetric. The first encoder layer consists of fully connected dense layer of size *(2 X HOG_vector)*, where *HOG_vector* is the size of the input vector. The second encoder layer is a fully connected dense layer of size *HOG_vector* (i.e. half the size of the first encoder layer). The latent layer is another dense layer that is half the size of the second encoder layer. The decoder is implemented to be the reverse of the encoder i.e. first hidden layer is a dense layer of same size as the encoder's second layer, while the second layer of the decoder is a dense layer of the same size as the encoder's first layer. This autoencoder architecture can be illustrated as follows, assuming A is the size of the HOG vector input:

Input layer – [Dense (2A) – Dense (A)] – latent layer- [Dense(A) – Dense (2A)] – Output layer

Note that we applied the Leaky ReLU activation for all the layers and used batch normalization for optimized learning.

IV. EXPERIMENTS AND RESULTS

In this section we present the experiments undertaken to investigate the performance of Android botnet detection schemes implemented within the Bot-IMG framework as described in sections II and III. The Bot-IMG framework is developed using Python and has been built using several Python libraries including: Scikit-learn, Numpy, OpenCV, PIL, Scikit-image, Keras, Pandas and Seaborn.

We performed experiments to study the performance of several machine learning classifiers including: KNN, SVM, Random Forest (RF), Decision Trees (DT), Extra Trees (ET), and XGBoost (XGB). These were used to evaluate the efficacy of the standard HOG feature descriptor and the enhanced HOG-based schemes implemented in the Bot-IMG feature extraction engine. Furthermore, we investigate potential performance improvement using autoencoders, where we have used a neural network to train an encoder to provide a compact representation of the HOG-descriptor vectors and in turn use this to train the standard machine learning classifier.

In this study, we used the Android dataset obtained from [13], which is known as the ISCX botnet dataset. The ISCX dataset contains 1,929 botnet apps (from 14 different families shown in Table I) and has been used in previous works including [7]–[10], [14], [15]. An additional 2,500 benign apps were obtained from different categories of apps on the Google Play store and confirmed to be clean using VirusTotal. Thus, a total of 4,429 apps (1929 botnets and 2,500 clean apps) were used for the experiments reported in this section.

In order to measure model performance, we used the following metrics: Accuracy, precision, recall and F1-score. The metrics are defined as follows:

- Accuracy: Defined as the ratio between correctly predicted outcomes and the sum of all predictions. It is given by:
$$\frac{(TP+TN)}{(TP+TN+FP+FN)}$$
- Precision: All true positives divided by all positive predictions. i.e. was the model right when it predicted positive? Given by:
$$\frac{TP}{(TP+FP)}$$
- Recall: True positives divided by all actual positives. That is, how many positives did the model identify out of all possible positives? Given by:
$$\frac{TP}{(TP+FN)}$$
- F1-score: This is the weighted average of precision and recall, given by:
$$\frac{(2\times Recall\times Precision)}{(Recall+Precision)}$$

Where TP is true positives; FP is false positives; FN is false negatives, while TN is true negatives. After obtaining the metrics for each class, we present the weighted average of both classes in the Tables below. Except for Table V, where

TABLE II
CLASSIFIER PERFORMANCE WITH HOG DESCRIPTORS FROM RESIZED IMAGES (10-FOLD CROSS VALIDATION RESULTS)

	Accuracy	Precision	Recall	F1-score
XGBoost	0.892	0.895	0.892	0.891
Extra Tree	0.863	0.885	0.863	0.858
Random Forest	0.871	0.881	0.871	0.861
KNN	0.877	0.899	0.881	0.887
SVM	0.811	0.845	0.814	0.811
Decision Tree	0.773	0.819	0.768	0.773

TABLE III
CLASSIFIER PERFORMANCE WITH HOG DESCRIPTORS FROM SEGMENTED IMAGES (10-FOLD CROSS VALIDATION RESULTS)

	Accuracy	Precision	Recall	F1-score
XGBoost	0.927	0.927	0.927	0.926
Extra Tree	0.925	0.920	0.919	0.918
Random Forest	0.919	0.920	0.919	0.918
KNN	0.877	0.930	0.846	0.878
SVM	0.866	0.894	0.865	0.867
Decision Tree	0.835	0.868	0.834	0.836

train-test split is used, all other results are from 10-fold cross validation where the dataset is divided into 10 equal parts with 10% of the dataset held out for testing, while the models are trained from the remaining 90%. This is repeated until all of the 10 parts have been used for testing. The average of all 10 results is then taken to produce the final result.

A. Performance of machine learning classifiers with HOG descriptors from resized dex file images.

As mentioned earlier, Bot-IMG was used to extract HOG feature descriptors for training machine learning models to identify botnets from the dex file images. In this section we present the results of evaluating various machine learning classifiers trained with HOG descriptors from grayscale images created to represent the botnet and benign apps.

The HOG parameters were set as follows: image size = 128 x 64 pixels; number of orientations = 9; pixels/cell = 8 x 8; cells/block = 2 x 2. Thus, the extracted HOG descriptors had a length of 3,780. As the images created from the apps were of different sizes, each one was resized to 128 x 64 before the HOG algorithm was applied.

In Table II, the results of XGBoost, (XGB), Extra Trees (ET), Random Forest (RF), SVM, KNN, and Decision Tree (DT) are shown. From the table, it can be seen that the best result was obtained by the XGB classifier, which had the highest F1 score of 0.891, with 89.2% accuracy, 89.5% precision and 89.2% recall. The lowest F1 score was 0.773 obtained with DT, corresponding to 77.3% accuracy. Note, that we experimented with various values of pixels/cell and cells/block while resizing the original dex images to 128 x 64, 64 x 64, and 128 x 128 respectively. We observed that changing these parameter values did not result in an improvement of classification accuracies.

B. Performance of machine learning classifiers with HOG descriptors from segmented dex file images.

In this section, we present the results of the machine learning classifiers on the enhanced HOG-based scheme where the descriptors are extracted from segments of the image representing the application. Our enhanced HOG-based scheme divides an image of size S = Height x Width pixels into N fixed-sized segments of dimension L x B. The results presented are based on N = 5 and L = 128 and B = 64. Hence, each segment is an image of dimension 128 x 64 pixels, and with 5 segments, only 40,960 bytes (i.e. 40 kb) of the dex image is utilized in the scheme. That means that part of the dex file is effectively truncated to about 40 kb in size. For the dex files that are less than 40 kb in size, there will be empty portions of the overall vector with zero-value HOG parameters.

The HOG parameters of each segment are set as follows: image (segment) size = 128 x 64 pixels; Number of orientations = 9; pixels/cell = 6 x 6; cells/block = 1 x 1. Thus, the extracted HOG descriptors of each segment had a length of 1,890. The total length of the HOG descriptor for the 5 segments is 9,450. Training the machine learning classifiers with a HOG descriptor vector of 9,450 is inefficient and would lead to longer training times. Hence the HOG descriptor vector length is reduced to 2,500 by taking 500 HOG descriptors from each segment.

Table III, shows the results of the enhanced HOG-based scheme for botnet detection. These results are from 10-fold cross validation, which means that they are the average results of 10 different runs with different training and testing portions. From the table, XGB had an F1 score of 0.926 and accuracy of 92.7%. This is an improvement over the results in Table II, where the F1 score and accuracy were only 0.891 and 89.2% respectively. The accuracy of Extra Trees is 92.5%, also a significant improvement compared to Table II, where the accuracy is 86.3%. In fact, all the machine learning models performed better with our enhanced HOG scheme compared to using the standard HOG approach. The improved performance could be attributed to the fact that the enhanced HOG scheme avoids resizing the images, but uses segments that preserves original pixel information instead.

C. Performance of Autoencoder and machine learning classifiers with HOG descriptors from segmented dex file images.

Table IV shows the results of the enhanced HOG scheme where we have employed Autoencoders to derive a compact feature representation from the HOG descriptor feature vectors. As mentioned earlier, using Autoencoders to derive new features also potentially results in improved performance.

In Table IV, we present the results of 10-fold cross validation, while Table V presents the results of 80:20 training and test split randomly selected on the entire dataset. From Table IV, the F1 scores of XGB, ET, RF and KNN are 0.931, 0.931, 0.924, and 0.901 respectively. The overall accuracies of XGB, ET, RF and KNN were 93.1%, 93.1%, 92.4% and 90.1% respectively. From Table V, on the 80:20 train-test split, XGB achieves up to 0.950 F1 score and 95% classification

TABLE IV
CLASSIFIER PERFORMANCE WITH AUTOENCODER AND HOG DESCRIPTORS FROM SEGMENTED IMAGES (10-FOLD CROSS VALIDATION RESULTS)

	Accuracy	Precision	Recall	F1-score
XGBoost	0.931	0.931	0.931	0.931
Extra Tree	0.931	0.932	0.931	0.931
Random Forest	0.924	0.925	0.924	0.924
KNN	0.901	0.901	0.901	0.901
SVM	0.897	0.897	0.897	0.897
Decision Tree	0.861	0.862	0.861	0.861

TABLE V
CLASSIFIER PERFORMANCE WITH AUTOENCODER AND HOG DESCRIPTORS FROM SEGMENTED IMAGES (80:20 TRAIN-TEST SPLIT RESULTS)

	Accuracy	Precision	Recall	F1-score
XGBoost	0.950	0.950	0.940	0.950
Extra Tree	0.953	0.960	0.950	0.950
Random Forest	0.939	0.940	0.930	0.940
KNN	0.917	0.910	0.920	0.910
SVM	0.912	0.910	0.910	0.910
Decision Tree	0.867	0.860	0.870	0.870

accuracy, while ET also obtained 0.950 F1 score with 95.3% classification accuracy. We also recorded higher accuracies for the other classifiers, with the 80:20 train-test split.

These results show that training the machine learning classifiers with the features that the encoder derived from the enhanced HOG descriptors improves their performance, and hence their ability to correctly predict the class of an unknown application.

V. RELATED WORK

In [16] L. Nataraj used grayscale images to distinguish between different malware families. However, their approach was based on the time-consuming GIST algorithm to extract features from the images. [17] and [18] used local binary patterns (LBP) to extract grayscale image features, while [19] used Intensity, Wavelet and Gabor to extract grayscale image features. Y. Dai et al. [20] used memory dump files of malware to generate grayscale images and obtained 95.2% classification accuracy. Han et al. [21] reduced the dimensions of grayscale images by using entropy images and used the similarity between the entropy images to detect malware. Burnap et al. [22], derived RGB images from machine activity data of an executing program and achieved 93.76% classification accuracy in detecting malware. In [23], malware opcode sequences were converted into RGB images and used to train an SVM classifier, achieving 92.86% accuracy. The aforementioned works focused on Windows (PE) malware. By contrast, this paper is focused on using visualization and image-based approaches to detect Android botnets. Moreover, the approach studied within the Bot-IMG framework is based on Histogram of Oriented Gradients.

Several works have been published regarding Android botnet detection in the past few years. For example, Alothman and Rattadilok [14], used source code mining with machine learning for the detection of Android botnets. Their approach reverse engineers Android apps and decompiles the executable to Java source code, and then applies text mining and Natural Language processing techniques to extract features for training KNN, Na¨ıve Bayes, J48, SVM and Random Forest algorithms. Compared to text mining approach, the image-based detection of Android botnets that our Bot-IMG framework facilitates, is much less cumbersome. Converting dex files to images and then extracting features for ML classification is far more efficient than decompiling dex files and extracting text-based features using NLP techniques.

In [24], a real-time approach based on dynamic analysis is proposed. It uses features from strace, netflow, logcat, sysdump and tcpdump and analyses these in a cloud-based virtual environment to detect Android botnets. A major drawback of their approach is the ability of botnets to detect virtual environments and evade them. Moreover, it is more time-consuming to detect botnets using a cloud-based virtual environment compared to the static image-based approach.

Currently, the most popular approach to Android botnet detection is by using hand-crafted or hand-engineered features such as API-calls, permissions, intents, broadcast receivers, background services, commands, strings, etc. For example, [10] used a combination of permissions, MD5 signatures, broadcast receivers, and background services as features for building machine learning classifiers. They performed experiments on 1,400 botnet apps from the ISCX botnet dataset, together with 1,400 benign apps. They obtained up to 95.1% accuracy, 0.827 recall and 0.97 precision in their results. ABIS [25], is the Android Botnet Identification System proposed based on static and dynamic features consisting of API calls, permissions, and network traffic. In [26], DeDroid was proposed, also based on static features. DeDroid first identifies 'critical features' by observing the coding behaviour of a few known malware binaries having command and control features.

The hand-crafted approach taken by ABIS, DeDroid and the other works, are not sustainable in the long run. This is because, as the Android OS evolves, new features are added and old ones may be deprecated. Thus, an in-depth level of domain knowledge is needed to maintain detection systems built upon had-crafted features. On the other hand, the Bot-IMG approach of using images for classification and detection of botnets is not closely tied to OS platform evolution and can easily be updated without intimate knowledge of the domain.

Image-based detection with machine learning has been applied to Android botnet detection in a previous paper [15]. In their approach, the applications are represented as images that are constructed from a matrix that records the co-occurrence of permissions used within the application. CNN is used to classify the app into 'clean' or 'botnet' based on a model built using the images constructed from botnet apps from ISCX botnet dataset and benign apps from various sources. The drawback of the approach used in [15] is that it is based on permissions, which is also a property that evolves with

newer versions of the Android OS.

Unlike previous works on Android botnet detection, that are either based on text mining, virtual execution or hand-engineered static features, this paper presents a framework that utilizes images to enable a more effective way to detect Android botnets. The results obtained with a HOG-based scheme implemented within the Bot-IMG framework shows that the image-based method using machine learning is promising for Android botnet detection.

VI. Conclusion and Future Work

In this paper we presented a framework for image-based Android botnet detection called Bot-IMG. The framework enables Android apps to be represented in various forms as grayscale (or optionally colour images) for machine learning based detection. The utility of Bot-IMG was successfully demonstrated through empirical evaluation of an enhanced HOG-based scheme implemented within its feature extraction engine. The enhanced HOG-based scheme replaces image resizing, which was the cornerstone of the original HOG approach, with image segmentation. This makes it possible to extract HOG feature descriptors without loss of information. The results of the experiments with various machine learning classifiers show significant improvements enabled by the proposed HOG-based scheme implemented within Bot-IMG. In order to improve performance further, we employed Autoencoders to derive a reduced feature representation from the HOG feature vectors. This ultimately improved the performance of the enhanced HOG scheme, yielding an average accuracy of up to 93.1% with XGBoost and Extra Trees using 10-fold cross validation and 95.3% with Extra Trees when evaluated using an 80:20 train-test split.

Having validated the Bot-IMG framework in this paper, for future work, we plan to implement other types of feature descriptors within its feature extraction engine and compare their performance to that of the enhanced HOG-based scheme. Future work will also explore the use of deep learning with various image representation forms derived from the Bot-IMG framework.

Acknowledgements

This research is supported by the 2021 Cybersecurity research grant from the Cybersecurity Center at Prince Mohammad Bin Fahd University, Al-Khobar, Saudi Arabia.

References

[1] McAfee Mobile Threat Report Q1. [Online]. Available: https://www.mcafee.com/en-us/consumer-support/2020-mobile-threat-report.html [Last accessed: 1 Oct., 2021]

[2] B. Grill, M. Ruthven, and X. Zhao. Detecting and eliminating Chamois, a fraud botnet on Android. [Online]. Available: https://android-developers.googleblog.com/2017/03/detecting-and-eliminating-chamois-fraud.html [Last accessed: 1 Oct., 2021]

[3] C. Brook. Google Eliminates Android Adfraud Botnet Chamois. [Online]. Available: https://threatpost.com/google-eliminates-android-adfraud-botnet-chamois/124311/ [Last accessed: 1 Oct., 2021]

[4] F. Y. Rashid. Chamois: The big botnet you didn't hear about. [Online]. Available: https://duo.com/decipher/chamois-the-big-botnet-you-didnt-hear-about [Last accessed: 1 Oct., 2021]

[5] Akamai, *Credential Stuffing attack, State of the Internet (SOTI)*. Report, 2018.

[6] Imperva, *Mobile Bots: The Next Evolution of Bad Bots*. Report, 2019.

[7] S. Y. Yerima and M. K. Alzaylaee, "Mobile botnet detection: a deep learning approach using convolutional neural networks," in *International conference on Cyber Security and Protection of digital Services*, ser. (Cyber Security 2020). IEEE, 2020.

[8] S. Y. Yerima, M. Alzaylaee, A. Shajan, and P. Vinod, "Deep learning Techniques for Android Botnet detection," *Electronics*, vol. 10, no. 519, 2021.

[9] W. Hijawi, J. Alqatawna, and H. Faris, "Toward a detection framework for android botnet," in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, 2017, pp. 197–202.

[10] S. Anwar, J. M. Zain, Z. Inayat, R. U. Haq, A. Karim, and A. N. Jabir, "A static approach towards mobile botnet detection," in *2016 3rd International Conference on Electronic Design (ICED)*, 2016, pp. 563–567.

[11] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 886–893.

[12] T. Surasak, I. Takahiro, C. Cheng, C. Wang, and P. Sheng, "Histogram of oriented gradients for human detection in video," in *2018 5th International Conference on Business and Industrial Research (ICBIR)*, 2018, pp. 172–176.

[13] ISCX Android botnet dataset. [Online]. Available: https://www.unb.ca/cic/datasets/android-botnet.html [Last accessed: 1 Oct., 2021]

[14] B. Alothman and P. Rattadilok, "Android botnet detection: An integrated source code mining approach," in *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2017, pp. 111–115.

[15] S. Hojjatinia, S. Hamzenejadi, and H. Mohseni, "Android botnet detection using convolutional neural networks," in *2020 28th Iranian Conference on Electrical Engineering (ICEE)*, 2020, pp. 1–6.

[16] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *VizSec '11: Proceedings of the 8th International Symposium on Visualization for Cyber Security*, 2011, pp. 1–6.

[17] J.-S. Luo and D. C.-T. Lo, "Binary malware image classification using machine learning with local binary pattern," in *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 4664–4667.

[18] H. Yan, H. Zhou, and H. Zhang, "Automatic malware classification via pricolbp," *Chinese Journal of Electronics*, vol. 27, no. 4, pp. 852–859, 2018.

[19] K. Kancherla and S. Mukkamala, "Image visualization based malware detection," in *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, 2013, pp. 40–44.

[20] Y. Dai, H. Li, Y. Qian, and X. Lu, "A malware classification method based on memory dump grayscale image," *Digit. Investig.*, vol. 27, pp. 30–37, 2018.

[21] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," *International Journal of Information Security*, vol. 14, 2015.

[22] P. Burnap, R. French, F. Turner, and K. Jones, "Malware classification using self organising feature maps and machine activity data," *Computers & Security*, vol. 73, pp. 399–410, 2018.

[23] T. Wang and N. Xu, "Malware variants detection based on opcode image recognition in small training set," in *2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 2017, pp. 328–332.

[24] S. Jadhav, S. Dutia, K. Calangutkar, T. Oh, Y. H. Kim, and J. N. Kim, "Cloud-based android botnet malware detection system," in *2015 17th International Conference on Advanced Communication Technology (ICACT)*, 2015, pp. 347–352.

[25] C. Tansettanakorn, S. Thongprasit, S. Thamkongka, and V. Visoottiviseth, "ABIS: A prototype of Android Botnet Identification System," in *2016 Fifth ICT International Student Project Conference (ICT-ISPC)*, 2016, pp. 1–5.

[26] A. Karim, R. Salleh, and S. A. A. Shah, "Dedroid: A mobile botnet detection approach based on static analysis," in *2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, 2015, pp. 1327–1332.